Teaching Three Quality Assurance Techniques in Tandem – Lessons Learned

Cat Kutay, Muhammad Ali Babar
Empirical Software Engineering
National ICT Australia Ltd. and University of New South Wales, Australia
{cat.kutay,malibaba}@nicta.com.au

Abstract

This paper presents our experiences gained in teaching software quality to undergraduate computer science and engineering students at The University of New South Wales. We argue that increasing demand of high quality software makes it imperative to teach a wide variety of skills which are required to deliver quality product or design and implement a quality focused process. We taught three quality improvement techniques to give students a greater appreciation of the range of the techniques available, and their respective strengths and weaknesses. We found it challenging but rewarding to inculcate programming minded students with the skills required to deal with product and process quality related issues. We believe that the experiences gained from this exercise will be valuable to those interested in designing and offering software quality education and training at tertiary level.

1. Introduction

Quality is one of the most important issues in software development. In software development, quality aims to deliver products of consistent standard with controlled quality profiles in an effort to minimize production expenses [7, 10]. Therefore, software engineering community has developed several quality assurance techniques to evaluate and improve the quality of software products and processes. Software inspections (or technical reviews) [9] and software testing [10] are well-known approaches being taught and practiced for over twenty years. Recently, researchers and practitioners are advocating the needs of addressing the quality related issues at the architecture level [3].

The idea of predicting the quality of a software intensive system from a high level design description originated in Parnas's work on software modularization [23], and has recently emerged as an important quality assurance technique known as software architecture (SA) evaluation. A number of methods, such as Architecture Tradeoff Analysis Method (ATAM) [18], Software Architecture Analysis Method (SAAM) [17] and Architecture-Level Maintainability Analysis (ALMA) [22], have been developed to assess the potential of a chosen SA to deliver a system capable of fulfilling required quality requirements and to identify potential risks [20].

We have been teaching software quality to third year undergraduate students of software engineering, computer engineering and computer science degree programs at The University of New South Wales, Australia, using software inspections as a primary means of teaching software quality assurance concepts [25]. The main objective of our pedagogy has been to impart the skills required to improve the quality of process and product while fostering student's construction of their own knowledge [7, 10, 13].

Since students' experiences of software processes are minimal at this stage, we attempt to provide a learning environment that emulates some aspects of software development without overwhelming them with new situations. Our lectures are designed to explain theoretical concepts underpinning a particular method and our tutorials are used for hands-on training for these methods.

In the most recent offering of the course, we also incorporated software testing and SA evaluation approaches as quality assurance techniques. The aim was to equip the students with the skills required to measure and improve the quality of software at different stages of the development lifecycle. While each technique is considered quite distinct we also tied these together by discussing the effect on downstream software development effort and quality.

2. Background of Quality in Engineering

Engineering has a long history of the study of quality in production and this forms a basis with which to produce quality software professionals. In particular the CMM software process improvement framework has analogies in established engineering approaches :

1) Define the process and measure its performance related to engineering design [8].
2) Establish process control for quality in terms of identifying and correcting process tasks based on statistical process control [11].
3) Design systems with a view to ongoing improvement of the production process systems [26] through innovation based on evolutionary operation [5].

Proceedings of the Fifth International Conference on Quality Software (QSIC'05)
1550-6002/05 $20.00 © 2005 IEEE

However software processes do not relate to manufacturing processes in their implementation. For instance, in manufacturing the process and materials are consistent across individual production outputs and quality metrics are generally clearly defined and measurable. At the fine-grained level software products are all unique and software processes are continually varying. However, there is certain similarity within the software industry at a higher level of abstraction

2.1 Teaching quality assurance

Teaching quality assurance techniques at a tertiary level in academic environment presents unique challenges. Students in tertiary courses are adults and have developed certain sense of their own expertise, which reduces their ability to recognize the need for change in their approach to software development.

We believe that pedagogy style at the university level should be designed to provide students with general skills and concepts to solve problems within their professional domain rather than providing them with specific skills or tools [24]. Several learning approaches have been developed in different disciplines to foster learning including Problem Based Learning (PBL). To enable students to think about quality issues they need to confront problems relating to quality maintenance. Thus, we designed a course based on the a contrained PBL approach. This course was a mixture of simple practical problems and processes that lead to solving those problems.

3. Quality Assurance Techniques Taught

Software Architecture Evaluation is recognized as an effective quality assurance technique that helps discover potential risks and questionable design decisions [3, 6]. The software architecture community has developed several architecture evaluation methods [2] such as the Architecture Tradeoff Analysis Method (ATAM) [6] and Architecture-Level Maintainability Analysis (ALMA) [22]. Though there are differences among these methods [2], we have identified five common activities by comparing four main SA evaluation methods [1]. These five activities can make up a generic scenario-based software architecture evaluation process which we teach:

- Evaluation planning and preparation
- Explain architectural approaches
- Elicit quality sensitive scenarios
- Analyze architectural approaches
- Interpret and present results

Having introduced the students to various activities and logistics of SA evaluation process, our teaching and assessment exercises were mainly focused on one vital activity of SA evaluation, developing quality sensitive scenarios. We focused on the activity of developing scenario profiles for several reasons. Developing quality sensitive scenario profiles is considered the most important task as scenarios are the means of precisely defining the required quality requirements. Moreover, the accuracy of the results of SA evaluation exercise is largely dependent on the quality of the scenarios used [3, 4].

Table 1: Scenario framework used [3].

Elements	Brief Description
Stimulus	A condition that needs to be considered when it arrives at a system
Response	The activity undertaken after the arrival of the stimulus
Source of Stimulus	An entity (human, system, or any actuator) that generates the stimulus
Environment	A system's condition when a stimulus occurs, e.g., overloaded, running etc.
Stimulated Artifact	Some artifact that is stimulated; may be the whole system or a part of it.
Response measure	The response to the stimulus should be measurable in some fashion so that the requirement can be tested.

The SA community has also developed different frameworks for eliciting, structuring and classifying scenarios. For example, Lassing et. al. [21] proposed a two dimensional framework to elicit change scenarios and Bass et. al. [3] provided a six elements framework to elicit and structure scenarios as shown in Table 1. We taught the students to develop and structure quality sensitive scenarios using this framework and to prioritize them.

Software Inspection Process is aimed at detecting faults in an artifact by having it reviewed by technically competent teams of inspectors [9]. Software inspection can be used to detect defects in all sorts of software artifacts such as requirements documents, design specifications, code, test cases and others. Technical and business reports can also benefit from inspection [14]. There are many models of inspections such as [9, 12, 16] and several variations of them. We taught an inspection process that is based on the technique described in [16]. This process model consists of four phases: Inspection planning, Preparation for inspection, Inspection meeting, Post inspection activities.

Software Testing was taught as a third quality assurance technique. We emphasized the need for identifying appropriate testing strategies according to requirements of the functions that need to be tested. Our approach was aimed at imparting the students with the skills to design a testing process according to a particular development approach such as agile, spiral or waterfall.

4. Course Structure

The Total Quality Management (TQM) is offered in seven academic weeks. The main objective of the lecture was to teach the theoretical concepts underpinning the contemporary approaches to improve the software process and product quality and interdependency among different approaches. The concepts of quality assurance were presented each week in the order shown in

Table **3**.

We did not emphasize too much on segregating the development lifecycle in different stages to avoid the impression that quality assurance techniques being taught are tightly integrated with a particular software development. We focused on how different techniques (e.g. SA evaluation, inspection, and testing) are used to assess and control the quality of the artifacts created during different stages of software development lifecycle and how the quality of the artifacts created during subsequent activities is dependent upon the quality of the artifacts developed by the earlier activities of the development lifecycle.

The tutorials were designed to reinforcing the theoretical concepts taught in the lecture with practical exercises of using different techniques being taught in the course. For example, for software architecture evaluation, the students were required to create and prioritize quality sensitive scenarios for real world systems. The practical exercises focused on different quality aspects at each stage and on the processes used to study these are listed in Table 2. Also shown are the linkages from each quality method used in the course to the CMMI process descriptions.

Table 2 Sequence of Quality Metrics taught with mapping to tasks used to develop experience.

Quality Method	Quality Metric	Learning Scenario	CMMI Analogy
Process Description	Consistency	Requirements Process	Inception of Process
Templates and Checklists	Repeatability	Architecture Evaluation	Repeat Process
Process Design	Understand-ability and Completeness	Architecture Evaluation and Inspection Process	Define Process
Metrics	Controllability	Architecture Evaluation and Inspection Process	Manage Process
Documenting and Traceability	Correctness	Testing	Optimize Process

The next step was to provide a process that enabled students to see different level of the CMMI model as applied to their learning environment. The areas of practical learning are listed in Table 3 to illustrate how the task sequence used during the course matched to the CMMI process levels.

Table 3 Experiential learning tasks during course mapping learning process to CMMI process.

Wk	Practical tasks	Process Description	Verification Documentation	Process Management	Process Optimization
2	Requirements Engineering process case study	Analyze process for ability to be enacted	Example process with errors given.	Consider each task and if it is complete and consistent	CMMI Process description provided for validation
3,4	Architecture Evaluation	Develop scenarios using two stage process	Example scenarios & templates provided	Brainstorm & discuss quality sensitive scenarios using procedural roles	Scenario development framework to structure the scenarios and prioritize them.
5	Code Inspection case study	Analyze and role play Code Review Process	Process definition provided	Develop metrics and task for process	Tutor provided feedback on quality aspects
6	Testing case study	Discuss designing for testing, suitable test types and test termination decisions	Analyze case study and discuss pros & cons of various testing techniques	Tutors led the discussion guided by a checklist of the issues to be consider	Tutors provide feedback on issues not considered

Software architecture evaluation considers architectural decision against business goals, rather than technical specifications as covered by technical reviews [3, 6]. Students were taught how scenarios are used to precisely define the business goals and quality requirements and how to develop such scenarios.

There were three tutorial exercises involving scenario generation which were designed to provide the students with the opportunity to learn and improve their skills in brainstorming, structuring, and prioritizing quality sensitive scenarios individually and in groups of three to five members. The students were also asked to evaluate

their designs with quality metrics. This required them to think about what constitutes a good design, and how to measure this, or how to compare two designs. Thus, the students were made to think at a high level of abstraction about metrics that are suitable to measure the quality of an abstract design. The templates also enhanced the traceability of the design to requirements. The entire process emphasized the need to document design decision to enable future changes to be traced back to this original evaluation

Code inspection followed the SA evaluation as a quality assurance technique which can be applied on more concrete artifacts. During the tutorial, the students used a code review case study to analyze and role play a code inspection process. From our experience of teaching software code inspection [25], we have found that the inspection process is an ideal vehicle for students to carry out a small exercise of quality control, consider the objectives of the tasks that they are performing and different means of measuring the process being followed.

The students were given an inspection process description template and asked to consider the applicability of the given template to a code inspection. They were asked to step through different activities documented in the template and reason about the various ways of implementing those activities to review code. While students are usually familiar with reading code they usually have difficulty in managing the inspection process. Hence we decided to train the students in managing the inspection process rather than performing inspection itself. They were provided extensive guidelines on inspection management related issues in the course notes.

During the tutorial, we discussed the use of checklists and templates to support the inspection process. The students also learned how to measure and improve the process. The objective was to impart the skill and confidence of enacting a process through knowing the objectives of the process rather than simply knowing how to follow the process.

Software testing was the final quality assurance techniques taught in the course. The emphasis was on testing as a process that verifies the product against the requirements and specification (using the V-model of development), hence requiring testability in all stages of development. Since testing only locates failures the important aspect is how to set the scope and end points of testing. The tutorial exercise case study looked at developing Test Plans and considered the metrics used to evaluate testing. Testing is also a process that can be controlled, as well as a quality control process itself.
The testing case study looked at the following aspects: designing a testing process, selecting appropriate testing strategies, and identifying appropriate metrics to be used. In particular we focused on looking into different criteria that can be used as a testing stopping point. Stopping testing is a critical decision that affects both testing efficiency and product quality. This was to focus on the issues of positive and negative correlation between metrics and the effect this has on process improvement.

5. Assessment Components and Procedures

The assessment for the course consisted of four components: tutorial exercises, essay writing, discussion participation, and final written examination. We provided a range of assessments as this enabled students to express their learning in forms in which they were most skilled.

Also, as an example of peer reviews as a quality assurance technique, one of the tutorial exercises was re-marked by the students. They were given specific instruction on what types of errors or quality factors they should be looking for to grade a tutorial submission. There was negative marking for making an inaccurate assessment of the quality of other's work. The objective was to learn how to assess their peers' work and bench mark their own work with certain degree of objectively.

The discussion participation component was for initiating and promoting discussion on various topics of software quality posted by the lecturer. Marks were based on the relevancy, quality, and stimulation capability of the postings of each student.

6. Measures of Success

The measurements we collected from the course are described here, with proposals for improvements in future enactments:

1. Weekly tutors meetings enabled tutors to comment on which aspects of the course were easy for students to enact, and those aspects (such as the scenario template described in the following section) with which comprehension was difficult. These can be used to improve tutorial exercises for next session
2. Students developed scenarios in three consecutive weeks, enabling us to measure the improvement of their product over this period while considering affects of factors such as different techniques used each week and differing groups each week.
3. Students marked each other's submissions and were assessed for incorrect assessment. This should be done earlier in course and repeated.
4. Assignment marks were marked for their understanding of the process rather than their actual choice of process tasks and metrics. Assignments could be matched with tutorial exercises so that students enact in tutorials the process they finally discuss in their assignment enabling them to experience any problems

embodied in their process design and metrics.

5. Final exam enabled us to examine which aspects of measurement were not grasped by the majority of students. Analysis of the final exam results can assist in finding aspects of the course to change and make clear, or exercises to improve.
6. Course evaluation questions at the end of the session were collected. These questions could be focused more on the perceived success of particular aspects taught to improve the relevance of this analysis.

7. Lessons Learned

In this section, we report on some of the important lessons learned while teaching various techniques of software quality assurance. These comments are qualitative but based on the metrics discussed in the previous section. Our aim is to share our experiences with those who are interested in designing academic or professional training courses .

One of the main teaching tasks was to enable students to learn the skill of developing metrics as a measure of quality and how to utilize these to improve the process. Our experiences from analyzing several industrial measurement initiatives are that most them collect data on the basis of ease of recording the metrics, rather than the importance of the metrics in optimizing the process. We tried to inculcate the students with the skills of identify metrics aimed at improving a certain aspect of software quality assurance process such as software inspection.

Moreover, we found there are many practical examples, which could be scaled down for a tutorial exercise to demonstrate the process required to develop quality software. Each tutorial task on scenarios was designed with increasing level of complexity of evaluation metrics.

Based on the feedback received from the tutorial team and the students, it can be asserted that SA evaluation tutorials using scenarios provided an opportunity to cover a broad range of metrics. Students found scenarios quite useful in learning the concepts of characterizing and evaluating various quality requirements with respect to a proposed SA. For example, we can use scenarios that represent failure to examine availability and reliability, scenarios that represent change requests to analyze modifiability, scenarios that represent threats to assess security, or scenarios that represent ease of use to analyze usability. Moreover, scenarios are normally very concrete, enabling the user to understand their effect [3] in detail.

Developing concrete scenarios to characterize quality attributes needs domain knowledge and rigorous thinking. To facilitate this task, we provided students with templates to stimulate their thinking and guide the process. However, students found that learning to use the template was quite difficult and required highly developed skill to work at a higher level of abstraction. We attempted to address this issue by getting the students to brainstorm, refine, and structure scenarios using the framework in a two stage process as tutorial exercise. This two stage process of developing scenarios was more effective.

We conjecture that another reason for not being able to quickly learn the use of framework may be that many students were from non-English speaking backgrounds and found it difficult to come up with suitable text segments of a scenario and then provide a concrete scenario. Or this problem may be a reflection of users' lack of understanding of the framework. In the latter case, there might have been the need for more examples of using the framework [15, 19].

We have found that it is normally difficult to get product focused (e.g. programming) students to fully concentrate on the human processes involved in the delivery or usage of the product (e.g. software system). The course involved assignments that required the students to study various aspects of improving the different processes of people using software, rather than designing of software being used by those people. Our general observation was that students found it hard to focus on the process as opposed to the product they were using in the process. We provided simple examples of different processes without including any direct reference to software development. However, we found that most of the students found it quite problematic to think of a process without thinking of the product developed to support the process.

Our experience of teaching the peer review process by having one of the assignments marked by the students was quite positive. We provided the metrics used for marking prior to the release of the assignment. Students reported that this exercise helped them understand various aspects of the course.

Apart from the difficulties that students faced in changing their mind set from product to process, they also reported difficulty in fully comprehending the course structure and coping with the reading load involved. We realized that the course had a complex structure in order to cover both the software development stages and the CMMI process improvement stages. We had developed a study guide consisting of several research articles on different topics included in the course and their interdependencies and relating the reading material to the necessary concepts to be focused on during the course. However, the structure of the course and the amount of the readings material deserved a full semester course, rather than the half session course allowed for in the program.

8. Conclusions

Most of the students of computing related degrees consider non-programming courses such as TQM and

project management to be easy options. In fact such courses usually prove quite difficult for programming minded student. A good grasp of the issues involved in software quality, process and product need a good understanding of the users of the products, the processes supported by the products and the processes of developing those products. It is often a challenging task for students to separate quantitative aspects of the problem and focus on the qualitative aspects of a project.

Thus, there is a need to teach technically-oriented student how to think at different levels of abstraction so as to relate their learning to the real life projects they are expected to be working on. We also conclude that instead of teaching the whole process of a particular technique to students who have little industrial experience, it may be more valuable to identify those tasks of the process, which are relatively easy to learn with the current level of experience and skills. We tried to achieve this by focusing on developing quality sensitive scenarios instead of the whole process of SA evaluation.

Students in computer-related degrees should be skilled not only in analytical and quantitative thinking but also in conceptual and qualitative reasoning. These skills are developed in dealing with ill–defined problems, which characterize most software development projects. We believe that students should be trained in addressing complex real world problems in the face of uncertain and evolving requirements and dynamic business processes. We find that existing curriculums of most of the computing-related degrees are not addressing this issue.

Acknowledgement: The authors would like to thank all the staff of the Total Quality Management course for their efforts and contributions.

9. References

[1] Ali-Babar, M. and I. Gorton. Comparison of Scenario-Based Software Architecture Evaluation Methods. Proc of the 1st Asia-Pacific Workshop on Software Architecture and Component Technologies. 2004. Busan, South Korea.
[2] Ali-Babar, M., et al. A Framework for Classifying and Comparing Software Architecture Evaluation Methods. Proc of the 15th Australian Software Engineering Conference. 2004. Melbourne, Australia.
[3] Bass, L., et al., Software Architecture in Practice. 2 ed. 2003: Addison-Wesley.
[4] Bengtsson, P. and J. Bosch, An Experiment on Creating Scenario Profiles for Software Change, Tech Report HK-R-RES-99/6-SE, University of Karlskrona/Ronneby, Sweden, 1999
[5] Box, G. and N. Draper, Evolutionary Operation: A Statistical Method for Process Improvement. 1969: John Wiley & Sons.
[6] Clements, P., et al., Evaluating Software Architectures: Methods and Case Studies. 2002: Addison-Wesley.
[7] Clements, P.C., ed. Constructing Superior Software. ed. S.Q.I. Series. 2000, Macmillan Technical Publishing, U.S.A.
[8] Diaz-Herreral, J.L. Engineering Design for Software: On Defining the Software Engineering Profession. Proc of the Proc. of the Frontiers in Education. 2001. Reno, Nevada, USA.
[9] Fagan, M.E., Design and Code Inspections to Reduce Errors in Program Development. IBM Systems Journal, 1976. **15**(3): p. 182-211.
[10] Galin, D., Software Quality Assurance From theory to implementation. 2004: Addison-Wesley.
[11] Gardiner, J.S. and D.C. Montgomery, Using Statistical Control of Charts for Software Quality. Quality and Reliability Engineering International, 1987. **3**: p. 40-43.
[12] Gilb, T. and D. Graham, Software Inspection. 1993: Addison-Wesley Professional.
[13] Glasersfeld, V., Cognition, Construction of Knowledge, and Teaching. SYNTHESE - Special Issue on Education, 1989. **80**(1): p. 121-140.
[14] Grady, R.B. and T.V. Slack, Key lessons in achieving widespread inspection use. Software, IEEE, 1994. **11**(4): p. 46-57.
[15] Hoffman, D. and P. Strooper, API Documentation with Executable Examples. Journal of Systems & Software, 2003. **66**(2): p. 143-156.
[16] Humphrey, W.S., Managing the Software Process. 1989: Addison-Wesley Professional.
[17] Kazman, R., et al. SAAM: A Method for Analyzing the Properties of Software Architectures. Proc of the 16th Int'l Conf. of Software Eng. 1994.
[18] Kazman, R., et al. Experience with Performing Architecture Tradoff Analysis. Proc of the Proc. of the 21th International Conference on Software Engineering. 1999. New York, USA: ACM Press.
[19] Kinshuk, T.L. Application of Learning Styles Adaptivity in Mobile Learning Environments. Proc of the Proc. of the 3rd Pan-Commonwealth Forum on Open Learning. 2004.
[20] Lassing, N., et al. The goal of software architecture analysis: Confidence building or risk assessment. Proc of the Proceedings of First BeNeLux conference on software architecture. 1999.
[21] Lassing, N., et al. On Software Architecture Analysis of Flexibility, Complexity of Changes: Size isn't Everything. Proc of the Proc. of 2nd Nordic Software Architecture Workshop. 1999.
[22] Lassing, N., et al., Experience with ALMA: Architecture-Level Modifiability Analysis. Journal of Systems and Software, 2002. **61**(1): p. 47-57.
[23] Parnas, D.L., On the Criteria To Be Used in Decomposing Systems into Modules. Communication of the ACM, 1972. **15**(12): p. 1053-1058.
[24] Schon, D.A., The Reflective Practitioner: How Professionals Think in Action. 1983, New York: Basic Books, Inc.
[25] Stalhane, T., et al. Teaching the Process of Code Review. Proc of the Australian Software Engineering Conference. 2004. Melbourne, Australia.
[26] Turner, W.C., et al., Introduction to Industrial and Systems Engineering. 2nd ed. 1987, Englewood Cliffs, N.J.: Prentice-Hall.